MURPHI BUSTS AN ALTITUDE: A MURPHI ANALYSIS OF AN AUTOMATION SURPRISE

Everett Palmer, NASA Ames Research Center, Moffett Field, California

Introduction

In training and during operations, users of automatic systems form expectations of how automatic systems respond to their control inputs and to environmental disturbances. These expectations form the basis for what can called the operator's "mental model" of the system. An "automation surprise" is said to occur when the automation behaves in a manner different from what the operator expects. A requirement for a properly functioning human-machine system is that the human operator have good situation awareness. A key component of an operator's situation awareness is knowing how the machine will behave in the near future. Automation surprises are situations in which this system requirement has failed. In this paper, the modeling language -- Murphi -- is used to model and analyze an automation surprise in which a flight crew, using the autopilot, climbs above their cleared altitude during a full mission flight simulation.

Murphi is a system description language and model checker developed by software engineers to formally evaluate behavioral requirements for concurrent software processes [2]. Formal methods allow designers to state and test the validity of general requirements such as, "the autoflight system will never fly the aircraft through the altitude set in the altitude window." A rule-based model of the autopilot system and the pilot was developed. Murphi was then used to automatically check the validity of the above requirement for a model of the pilot-autopilot-aircraft system. The requirement failed for the same sequence of human and machine events that were recorded in the altitude bust incident. The Murphi model was then modified to explore possible procedural and mode logic fixes to reduce the likelihood of this type of breakdown in the human-machine system.

Other applications of formal software engineering methods to this and other problems of automation surprises and mode confusion are described in [4, 5, 6 & 8].

Key Incident Events During the Altitude Bust

The incident took place during a high workload flight segment. The crew had just made a missed approach and had been cleared to climb to 5,000 feet. The captain correctly set the target altitude of 5,000 feet in the autopilot's "Mode Control Panel (MCP)" and engaged the "Vertical Speed" pitch autopilot mode to climb at about 2,000 feet per minute. The autopilot was armed to automatically capture the cleared altitude. At about 4,000 feet, the pilot changed the pitch mode to IAS. Altitude capture was still armed. Three seconds later the pitch mode automatically changed to "Altitude Capture." This pitch mode smoothly reduces the aircraft's vertical velocity until the aircraft is flying level at the target altitude. This mode change also disarms the altitude capture. A fraction of a second later the captain adjusted the vertical speed wheel. This caused the autopilot to switch to the Vertical Speed mode but now altitude

This work was supported by the Human-Automation Integration Project of NASA's Aerospace Operations System R & T Base Program.

capture was no longer armed and the autopilot was set to climb with no altitude target. Neither pilot took any action until the aircraft was climbing through the cleared altitude. As the aircraft simulator climbed through the cleared altitude of 5,000 feet, the captain exclaimed, "Five thousand. Oops, it didn't arm." indicating both a misunderstanding of what happened and providing evidence of an automation surprise. This altitude bust incident is described in more detail in [7].

Figure 1 is a finite state diagram of the pitch autopilot modes. This logic diagram was reverse engineered from the flight manual and a video tape of the flight mode annunciator. It is undoubtedly is a simplification of the actual system but does capture the automation behavior necessary to understand this incident.

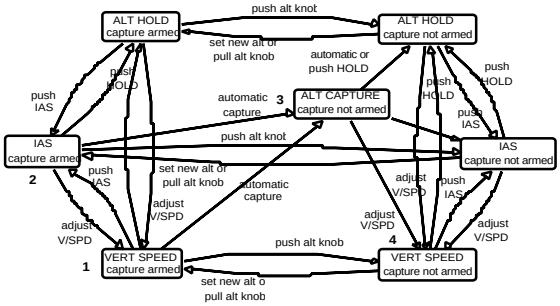

Figure 1. Pitch Autopilot Altitude Change and Capture Modes. The numbers show the sequence of state changes that occurred during the incident.

This particular pilot error can be described as a mode error. If the pilot had adjusted the vertical speed wheel a few seconds earlier, the result would have been only to change to vertical speed mode at a slightly different climb rate with the aircraft still automatically leveling at the target altitude. However, adjusting the vertical speed wheel after the change to the altitude capture pitch mode results in the aircraft continuing to climb without an altitude limit. A mode error occurs when an operator performs an action appropriate for an expected mode but the system is in a different mode for which the action has undesirable consequences. The automatic mode change to the altitude capture mode that preceded the pilot's mode error can be called a "silent" mode change. It is "silent" because it occurs automatically with the only indication being a visual change in the Flight Mode Annunciator.

A Murphi Model of the Human-Machine System

A Murphi model specifies the state variables of the system, their values and rules describing the conditions under which these states variables can change.

The system states and their possible values are defined as:

- Altitude {below_target, at_target, above_target}
- V_Speed: {level, climbing}
- In_capture_zone: {true, false}
- Pitch_mode: {vert_speed, alt_cap, alt_hold}
- Capture_armed: {true, false}

The following three rules describe possible pilot actions. Note that these rules specify only one sequential dependency, the pilot must raise the target altitude before adjusting the vertical speed. With this exception, pilot actions can occur at any time and in any order. Modeling possible pilot actions as opposed to just nominal pilot actions allows Murphi to automatically check system behavior for any sequence of pilot actions.

```
rule "raise_target_altitude"
begin
  altitude := below_target;
  capture_armed := true;
end;

rule "adjust_vertical_speed"
begin
  if (altitude = below_target) then
    pitch_mode := vert_speed;
    v_speed := climbing;
  endif;
```

```
end;

rule "push_alt_hold"
begin
  pitch_mode := alt_hold;
  v_speed := level;
  in_capture_zone := false;
end;
```

The following four rules describe the automatic behavior of the pitch autopilot.

```
rule "in_capture_zone"
begin
  if (altitude = below_target
      & v_speed = climbing) then
    in_capture_zone := true;
    if capture_armed then
       pitch_mode := alt_cap;
       capture_armed := false;
    endif;
  endif;
end;

rule "at_altitude_target"
begin
  if (pitch_mode = alt_cap
      & altitude = below_target
      & v_speed = climbing) then
    pitch_mode := alt_hold;
-- capture_armed := false;
    altitude := at_target;
    v_speed := level;
    in_capture_zone := false;
  endif;
end;

rule "almost_bust"
begin
  if (altitude = below_target
      & v_speed = climbing
      & pitch_mode = vert_speed
      & in_capture_zone) then
    altitude := at_target;;
  endif;
end;

rule "altitude_bust"
begin
  if (altitude = at_target
      & v_speed = climbing) then
    altitude := above_target;
    error "altitude_bust";
  endif;
end;
```

This model leaves out the "IAS" pitch mode. This simplification does not affect the important behavior for modeling this system. The model also only models the human-machine system during the climb. During a climb the aircraft should never climb above the target altitude. In this model, it is an error if the state variable "altitude" is ever equal to "above_target."

A Murphi model can be run in either simulation or verification mode. In simulation mode, Murphi selects at random among rules whose conditions are satisfied. In verification mode, Murphi performs an exhaustive search to determine if a sequence of rule firings exists that will take the system from the start state to an error state. Starting in level flight, Murphi identifies the following sequence of rule firings that leads to the aircraft being above its target altitude: 1) raise_target_altitude, 2) adjust_vertical_speed, 3) in_capture_zone, 4) adjust_vertical_speed, 5) almost_bust, and 6) altitude_bust. The key event (#4) leading to the error state is adjusting the vertical speed wheel after the autopilot automatically transitions to the altitude capture pitch mode. These events and the resulting state changes are shown in figure 2.

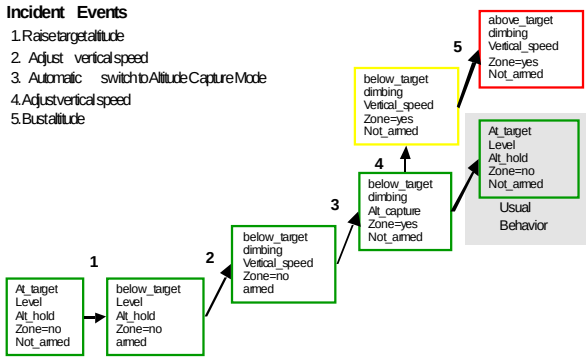

Figure 2. The sequence of state variable changes and human-machine events leading to the altitude bust.

We can now ask what went wrong and use Murphi to model and analyze possible procedural and autopilot mode logic changes

that could reduce the likelihood of this type of breakdown of this normally error-tolerant human-machine system.

What went wrong? ... What can be done?

In this incident the autopilot performed exactly as it was designed to perform. It is interesting to ask what it is about the pilot-autopilot-aircraft system design that makes the system prone to this type of automation assisted altitude bust.

Pilot error & procedural changes:

It is easy, though not very useful, to put the blame on pilot error. The flight operations manual contains a specific warning that adjusting the vertical speed while the aircraft is in altitude capture mode can lead to an altitude deviation. Still it is possible to use Murphi to evaluate two procedural fixes. One procedural change is for the pilot to check the Flight Mode Annunciator after any change to the autopilot controls and if altitude capture is not armed then rearm it. The following Murphi pilot rule captures this idea:

```
rule "rearm_altitude_capture"
begin
  if (altitude = below_target
      & pitch_mode = vert_speed
      & capture_armed = false) then
    capture_armed := true;
  endif;
end;
```

This rule prevents the human-machine system from busting the target altitude if the "pilot" remembers to check if altitude capture is armed or not. Another procedural change is to modify the "adjust_vertical_speed" rule to rearm altitude capture whenever the vertical speed is adjusted. A Murphi verification analysis indicates that this change would be effective but the problem in a real system would be in training pilots to reliably perform this action which is almost never required.

Autopilot mode logic changes:

Murphi can also be used to explore possible changes to the autopilot logic. We can impose a design requirement that no sequence of pilot inputs to the autopilot will result in the autopilot flying the aircraft through the target altitude. Our objective is to modify the mode logic so that the pilot can change a subgoal, the vertical velocity at which the aircraft is climbing, with out inadvertently canceling the main goal of capturing the cleared altitude.

One way to accomplish this to modify the system so that altitude capture remains armed until the target altitude is reached. In the Murphi model this can be done by moving the action "capture_armed := false;" from the rule "in_capture_zone" to the rule "at_target_altitude." The single rule for adjusting vertical speed is split into the following two rules. The first rule models the case where the new vertical speed leaves the aircraft in the capture zone. The autopilot logic stays in the altitude capture pitch mode. The second rule covers the case where pilot lowers the vertical velocity to a value that result in the aircraft leaving the capture zone. The logic changes to the vertical speed pitch mode.

```
rule "adjust_vertical_speed_1"
begin
  if (altitude = below_target) then
  pitch_mode := vert_speed;
  v_speed := climbing;
   if (in_capture_zone & capture_armed) then
       pitch_mode := alt_cap;
    endif;
  endif;
end;
```

```
-- The pilot adjusts vertical_speed to a lower
-- value that results in not meeting the criteria
-- for being in the capture zone.

rule "adjust_vertical_speed_2"
begin
  if (altitude = below_target) then
  pitch_mode := vert_speed;
  v_speed := climbing;
   if in_capture_zone then
      in_capture_zone := false;
    endif;
  endif;
end;
```

Running Murphi in verification mode with these modifications to the altitude capture logic shows that the pilot can adjust the aircraft's vertical velocity at any time without risking that the autopilot will climb above the target altitude.

The differences between the original and the revised design highlight an interesting design tradeoff. The revised design makes it more difficult for the pilot to inadvertently climb above an assigned altitude but it also makes it more difficult for the pilot to intentionally override the capture maneuver and continue climbing. With the revised design in order to override the capture, the pilot has to either raise the target altitude and select a climb mode. This design has the advantage of requiring an explicit pilot action to change the primary goal of climbing to a target altitude. The disadvantage is that more actions are required.

Discussion

It is difficult to know exactly what a pilot's expectations of the autopilot system should be but a reasonable assumption is that the pilot will expect that the autopilot will attempt to achieve the goal that it has been set up to perform. In this case, the pilot having correctly set the autopilot up to climb to the target altitude of 5,000 feet could be assumed to have the expectation that the autopilot would perform this task unless specifically instructed otherwise. The pilot did adjust the way in which the autopilot should perform the climb but never explicitly changed the goal of climbing to the target altitude.

The philosopher, Daniel Dennett, uses the term "intentional stance" to refer to this approach to predicting the behavior of people or man-made systems. Taking an "intentional stance" is a strategy for predicting the behavior of people or machines that we assume have been rationally designed. The autopilot is designed to perform specific tasks such as climbing to and maintaining a specified altitude. A reasonable expectation for the pilot is that the system will attempt to perform the task that has specified. Changing a subgoal, the vertical speed, should not negate the primary goal of climbing to the target altitude.

Designing autopilot flight mode annunciator displays to more explicitly indicate the intentions of the automation has been shown to result in more accurate predictions of machine behavior by pilots [3].

Concluding Remarks

The Murphi modeling system appears promising as a way to analyze the possible behavior of human-machine systems and to design automation behavior that is less likely to surprise it's human operators.

Murphi provided a straightforward language to model possible machine and human behavior. The Murphi language has the advantage of allowing minimal constraints on pilot behavior. This allows the implications of all possible, not just nominal, pilot behavior to be explored. For example, the model of the pilot put no restrictions on when or how often the pilot could change the target altitude or adjust the vertical velocity.

This autopilot design problem is a good demonstration that solving a human factors problems may not be just a matter of more training, better procedures or even better interface design. In this case, the best solution appears to be to redesign the autopilot logic so that the system behavior conforms to a reasonable operator expectations.

References

[1] Daniel C. Dennett, The intentional stance, MIT Press, 1987.

[2] David L. Dill. The Murphi verification system. In Rajeev Alur and Thomas A. Henzinger, editors, Computer-Aided Verification, CAV '96, volume 1102 of Lecture Notes in Computer Science, pp. 390-393, New Brunswick, NJ, July/August 1996. Springer-Verlag. Information on Murphi is available at http://sprout. stanford.edu/dill/murphi.html.

[3] Michael Feary, McCrobie, D., Alkin, M., Sherry, L., Polson, P., Palmer, E., & McQuinn (1998). Aiding vertical guidance understanding. NASA TM 1998-112217, Ames Research Center, Moffett Field, CA. Available at http://olias.arc.nasa.gov/~feary /aiding_vg/aiding_vg_understanding.html

[4] Denis Javaux and Peter G. Polson. A method for pre-dicting errors when interacting with finite state ma-chines. In Denis Javaux, editor. Proceedings of the 3rd Work-shop on Human Error, Safety, and System Development (HESSD'99), University of Liege, Belgium, June 1999.

[5] Nancy G. Leveson and Everett Palmer. Designing automation to reduce operator errors. In Proceedings of the IEEE Systems, Man, and Cybernetics Conference, October 1997. Available at http://www.cs. washington.edu/research/projects/safety/ www/papers/smc.ps.

[6] Nancy G. Leveson, L. Denise Pinnel, Sean David Sandys, Shuichi Koga, and Jon Damon Rees. Analyzing software specifications for mode confusion potential. In C. W. Johnson, editor, Proceedings of a Workshop on Human Error and System Development, pp. 132–146, Glasgow, Scotland, March 1997. Glasgow Accident Analysis Group, technical report GAAG-TR-97-2. Available at http://www.cs. washington.edu/research/projects/safety/ www/ papers/glascow.ps.

[7] Everett Palmer, "Oop's, it didn't arm." A case study of two automation surprises. Proceedings of the Eighth International Symposium on Aviation Psychology, pp. 227-232, Columbus, Ohio, 1995. Available at http://olias.arc.nasa.gov/~ev/ OSU95_Oops /PalmerOops.html.

[8] John Rushby, Using model checking to help discover mode confusions and other automation surprises, Proceedings of the 3rd Workshop on Human Error, Safety, and System Development, University of Liege, Belgium, 1999.